Alpha Dithering

Peter L. Williams*, Randall J. Frank, and Eric C. LaMar

Lawrence Livermore National Laboratory
Livermore, CA 94551

Abstract

This paper describes and analyzes a technique for accurately specifying small values of alpha that would normally not be possible due to the limited precision (number of bits) allowed for the alpha channel in graphics hardware.

1 Introduction

Graphics hardware generally allows an *alpha* value to be specified which is used to blend the color of an incoming fragment with the color of a pixel stored in the framebuffer. This blending technique is often used to render semitransparent objects. However, current graphics hardware offers limited precision, typically 8 bits, for the specification of alpha values, thus limiting these values to integers in the range $[0, 255]$. This can be a problem — for example in volume rendering or in creating volumetric effects such as rendering fire and fog. Then the per-pixel blending equation is: $NewColor = (\alpha * IncomingColor) + (1 - \alpha) * ExistingColor$.

When volume rendering using slices, alpha values are calculated as $\alpha = f(d, \tau)$, where d is the distance between slices and τ is the optical density of the material. Generally, the majority of alpha values are quite small

*L-560, LLNL, 7000 East Ave, Livermore, CA 94551, 925-422-3832, plw@llnl.gov

in order to allow sufficient transparency to give a penetrating view of the object. An image may be generated using progressive refinement: initially a few slices are rendered, then as time permits the resolution of the image is improved by rendering it with more slices. However, when the number of slices is increased, the alpha values need to be decreased since the distance between slices decreases. If the original alpha values were small, the new (even smaller) values may lack significant precision to register. For example, consider original alpha values of 1 and 3, which are reduced by one-half, so the new values become 1.5 and 0.5; converting them to integers will result in values of either 2 and 1, or 1 and 0. Neither of these sets of values results in semitransparent images that are comparable with the original image. See Figure 2. What we really need is some way to preserve the precision of the new alpha values.

In the next section, we discuss a software technique for dithering the specification of low alpha values that overcomes the hardware limitation described above and results in improved volume rendering results using progressive refinement. In Section 3, we analyze the effects of the dithering method and show some comparative images with and without dithering.

2 The Alpha Dithering Technique

In volume rendering, typically the colors and alphas are calculated from a user-defined transfer function $g(s) = (R, G, B, A)$, where s is the scalar field value being volume rendered. For efficiency, the transfer function is often stored in a table as a one dimensional texture map, and the scalar field value is specified as a texture coordinate.

To preserve the accuracy of small alpha values, in the face of limited hardware precision as described in Section 1, we dither small alpha values over a set of P slices. For every set of P slices, we render $P * (\alpha - \lfloor\alpha\rfloor)$ slices with an alpha value of $\lceil\alpha\rceil$ and $P * (1 - (\alpha - \lfloor\alpha\rfloor))$ slices with an alpha value of $\lfloor\alpha\rfloor$. We call P the *period*.

So for example if we choose a period of 16 slices, and alpha is 1.25, then 12 slices are generated using an alpha of 1, and 4 slices using an alpha of 2. The alphas may be alternated in a regular pattern such as $\{2, 1, 1, 1, 2, 1, 1, 1, 2, 1, 1, 1, 2, 1, 1, 1\}$, or randomly. If alpha had been 1.3,

then there is no dither pattern with a period of 16 that will exactly simulate it, so we round down to the closest dither pattern, in this case the pattern for alpha of 1.25. The basic dither patterns are calculated once for a given number of slices and a given transfer function and then stored. Between each slice, the appropriate dither pattern is downloaded into texture memory.

Dithering is often not essential for large alpha values, therefore we set an alpha cutoff value above which we do not do dithering. For our application we need to accumulate alpha as well as color in the frame buffer — this allows us to later blend other objects with our image. Since the blending formula specified in the OpenGL standard does not support accumulating alpha *and* color blending, we need to premultiply the colors of the transfer function by alpha. In this case, we premultiply the colors by the undithered alpha and then dither the colors. The dithered premultiplied colors and the dithered alphas are downloaded into texture memory between each slice.

The effective range and precision of alpha can be extended by keeping alpha values in a table of floats, this allows an initial alpha value to be a noninteger, e.g. 0.33 or 2.5. Then between each slice, we first dither the values in the table of floats into a table of bytes and then download that.

3 Analysis and Examples of Dithering

Figure 1 shows the effect of dithering as a function of output intensity and alpha. The benefits of dithering can be clearly seen.

Figure 2 shows six volume rendered images of a scientific data set. The top images were generated using 128 slices, the middle pair using 275 slices, and lower ones using 600 slices. The left set of images was created without alpha dithering, the right set with alpha dithering using a regular pattern over a period of 16 slices. The alpha dithering cutoff value was set to 3. The image with 128 slices was chosen as the reference image and the initial alpha values set to give it the desired transparency.

It should be noted that, for low alpha values the dither pattern may contains zeros, then dithering may involve a trade-off in image quality.

On the one hand, increasing the number of slices can be expected to reveal more of the geometric structure in the image. Whereas, dithering, which will preserve low alpha (diaphanous) areas of the image, may mitigate to some extent the effect of the increased number of slices. For example, if alpha is 0.5, then the dither pattern might be $\{1, 0, 1, 0, ..\}$, in which case portions of slices 2, 4, .. may be transparent.

4 Conclusion and Extensions

In our work, we apply these dithering techniques to volume rendering slices of a 3D data set of (unstructured) meshed polyhedra — so each slice consists of a set of polygons. However, our dithering procedure is applicable to any hardware compositing technique, such as texture-based volume rendering (e.g. using static or paletted textures, or pixel programs.)

To avoid the overhead of downloading a new texture map between each slice, it is possible to place a series of 1D textures into a 2D texture and then download it once. Each 1D texture map specifies one of the desired alpha dither patterns. This requires one additional texture coordinate for the data in each slice to indicate which texture map to use. This procedure can be made more efficient by the use of a texture matrix or automatic texture-coordinate generation.

ACKNOWLEDGEMENTS

We are grateful to Nelson Max for helpful conversations in regard to dithering. This work was performed under the auspices of the U.S. Department of Energy by the University of California, Lawrence Livermore National Laboratory under contract number W-7405-Eng-48.

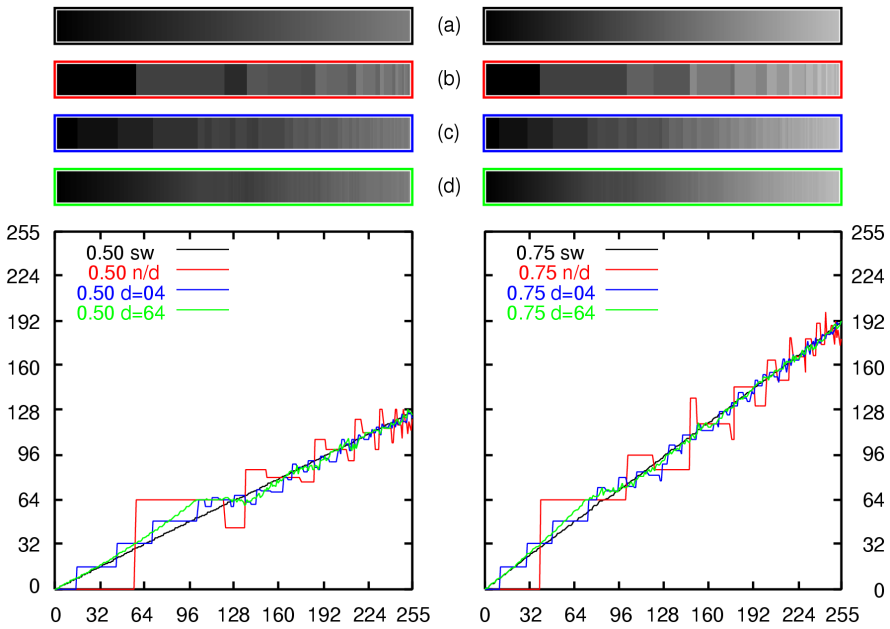

Figure 1: The plots compare rendering 64 slices using (a) software alpha blending (black line) with: 8 bit hardware blending using (b) no dithering (red); (c) dithering with a period of 4 slices (blue); and (d) dithering with a period of 64 slices (green). Intensity is shown vertically, alpha values horizontally. The figure on the left strives for a linear ramp to 50% intensity, the one on the right to 75% intensity. The four images above the plot were created using these methods respectively, from top to bottom.

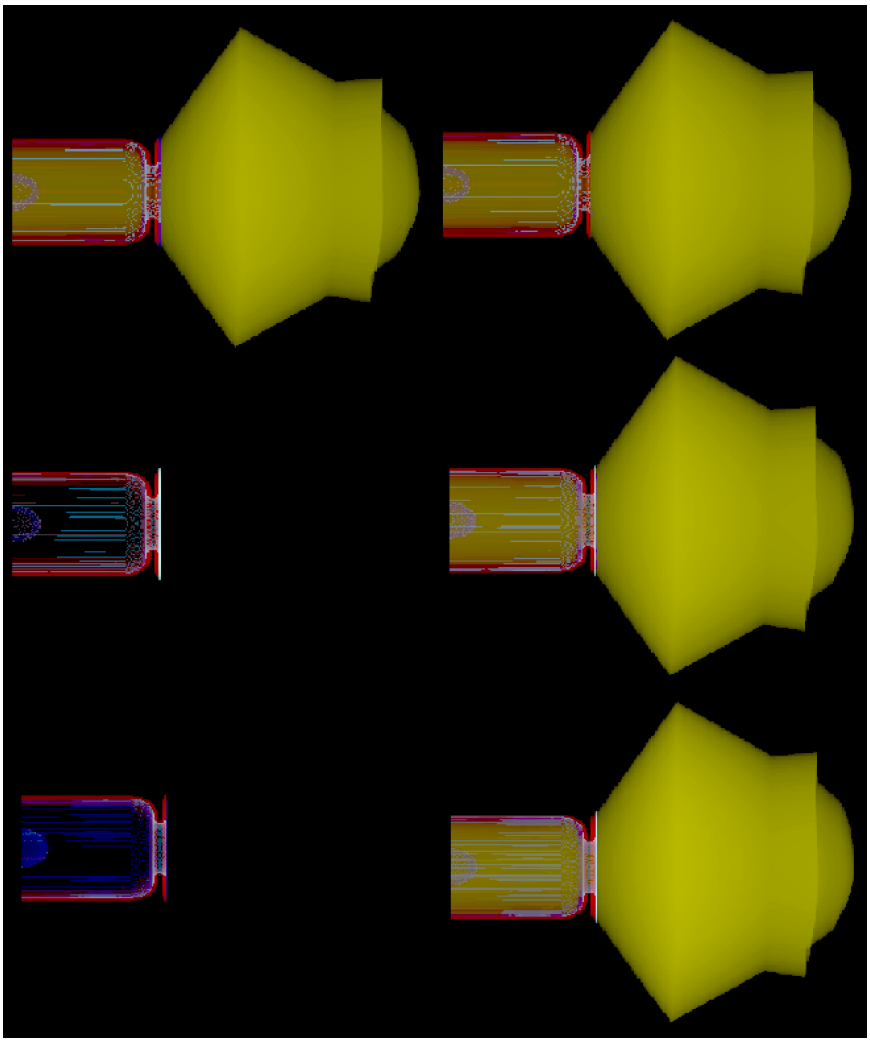

Figure 2: Volume rendered images of a scientific data set generated with, from top to bottom, 128 slices, 275 slices, and 600 slices. The images on the left were generated without alpha dithering, those on the right with alpha dithering. As the number of slices increases, the alpha values decrease to the point where small alpha values lose precision unless dithering is used.